Domain-Specific Software Architecture Engineering Process Guidelines ADAGE-IBM-92-02B Version 2.1 (1993) (Make Corrections) (8 citations)

Will Tracz, Lou Coglianese

ACM SIGSOFT Software Engineering Notes

Home/Search Bookmark Context Related

View or download:
owego.com/dssa/lmdocs/IBM9202.ps
Cached: PS.gz PS PDF DjVu Image Update Help

From: owego.com/dssa/lmdocs/lmdocs (more)
Homepages: W.Tracz [2] L.Coglianese
HPSearch (Update Links)

(Enter summary)

Rate this article: 1 2 3 4 5 (best)
Comment on this article

Abstract: "In order to reuse software, there needs to be software to reuse[12]." One of the dilemmas that has prevented software developers from reusing software is the lack of software artifacts to use or the existence of artifacts that are difficult to integrate. Domain-Specific Software Architectures (DSSAs) have been proposed [7] in order to address these issues. A DSSA not only provides a framework for reusable software components to fit into, but captures the design rationale and provides for a... (Update)

Context of citations to this paper: **More**

...data within a specified period of time from the occurrence of an event. **Application designers often impose these implementation constraints[10] to satisfy the needs of interfaces with other systems.** Any component in the architecture may have this requirement imposed. The most...

.... **[HC91] Since then, Domain Analysis has been associated with the development of reusable software components [Gom91] HC91] Tra92] TCY93] Some reuse is apparent while other reuse is more difficult to identify.** For example, it is easy to imagine a collection of reusable...

Cited by: **More**
Reconfigurable Architectures for Mixed-Initiative Planning and.. - Becker (1998) (Correct)
Aviation System Analysis Capability Executive.. - Roberts, Villani.. (1999) (Correct)
Components, Frameworks, Patterns - Johnson (1997) (Correct)

Similar documents (at the sentence level):
19.7%: A Domain-Specific Software Architecture Engineering.. - Tracz, Coglianese, Young (1993) (Correct)

Active bibliography (related documents): **More All**
1.4: Domain-Specific Software Architecture Engineering Process .. - Tracz, Coglianese, Young (1993) (Correct)
0.3: DSSA-ADAGE Design Records - Tracz, Shafer, Coglianese (1994) (Correct)
0.2: Characteristics of Higher-level Languages for Software.. - Shaw, Garlan (1994) (Correct)

Similar documents based on text: **More All**
0.3: Architecture Component Relationships for the DSSA-ADAGE Project - Coglianese (1993) (Correct)
0.3: ADAGE DSSA Components and Architecture Description in LILEANNA - Tracz, Coglianese (Correct)
0.2: A Domain-Specific Software Architecture for a Class of.. - Hayes-Roth (1994) (Correct)

Related documents from co-citation: **More All**
2: Feature-oriented domain analysis: Feasibility study (context) - Kang, Cohen et al. - 1990
2: Construction of File Management Systems from Software Components (context) - Batory, Barnett et al. - 1989
2: DSSA frequently asked questions (context) - Tracz - 1994

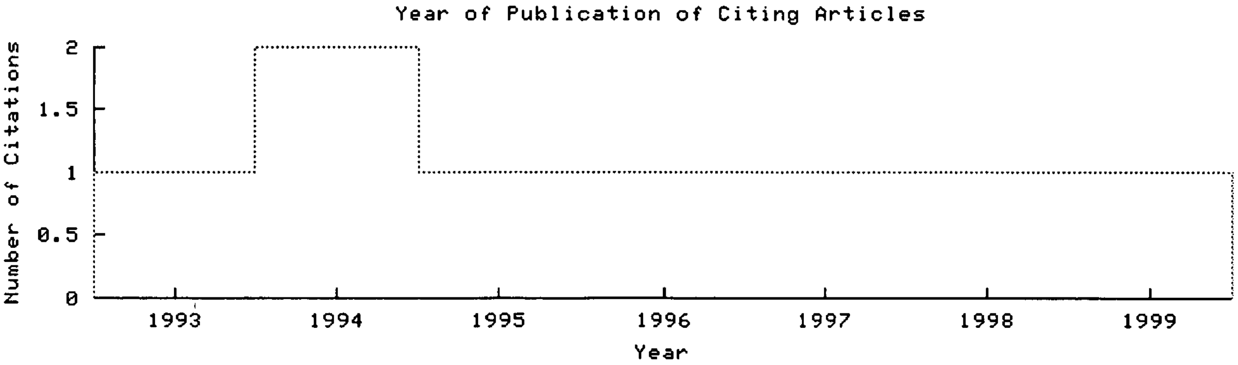
BibTeX entry: **(Update)**

Tracz, W., Coglianese, L., Young, P. A domain-specific software architecture engineering process outline. SIGSOFT Software Engineering Notes, 18, 40-49, 1993. http://citeseer.nj.nec.com/tracz93domainspecific.html More

```
@article{ tracz93domainspecific,
    author = "W. Tracz and L. Coglianese and P. Young",
    title = "A Domain-Specific Software Architecture Engineering Process Outline",
    journal = "ACM SIGSOFT Software Engineering Notes",
    volume = "18",
    number = "2",
    pages = "40-49",
    year = "1993",
    url = "citeseer.nj.nec.com/tracz93domainspecific.html" }
```

Citations (may not include all citations):

51 Software Components with Ada (context) - Booch - 1987
50 Feature-Oriented Domain Analysis (context) - Kang, Cohen et al. - 1990
46 Issues as Elements of Information Systems (context) - Kunz, Rittel - 1979
37 A Technique for Software Module Specification with Examples (context) - Parnas - 1972
33 Domain Analysis for Reusability (context) - Prieto-D - 1987
31 SADT Structured Analysis and Design Technique (context) - Marca, McGowan - 1988
10 Reuse Library Process Model (context) - Prieto-Diaz - 1991
8 A Conceptual Model for Megaprogramming (context) - Tracz - 1991
5 Presentation at ISTO Software Technology Community Meeting (context) - Mettala, Software et al. - 1990
4 Software Engineering Institute (context) - Lee, Paradigm et al. - 1988
4 ACM Software Engineering Notes (context) - Tracz, Maxims - 1988
3 RDD-100 Requirements Driven Design User's Guide (context) - Corporation - 1991



The graph only includes citing articles where the year of publication is known.

Documents on the same site (http://www.owego.com/dssa/lm-docs/lm-docs.html): More
A Domain-Specific Software Architecture Engineering.. - Tracz, Coglianese, Young (1993) (Correct)
DSSA-ADAGE Design Records - Tracz, Shafer, Coglianese (1994) (Correct)
ADAGE DSSA Components and Architecture Description in LILEANNA - Tracz, Coglianese (Correct)

Online articles have much greater impact More about CiteSeer Add search form to your site Submit documents Feedback

CiteSeer Find: requirements traceability and so Documents Citations

Searching for **requirements traceability and software design**

Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP

6 documents found. **Order: citations weighted by year.**

Formal Modelling and Simulation in the Development of a.. - Larsen Fitzgerald (1995) (Correct) (1 citation)
Ward &Mellor's extensions supported by **requirements traceability** tools)but one team additionally uses
resolved by recorded queries and answers. 2. **Software Design** The system design is given further detail
ftp.ifad.dk/pub/papers/anglo.ps.gz

Thoughts on Software Engineering Knowledge, and how to.. - Bagert, Barbacci.. (Correct)
issues in software architecture DES.ar.4 **Requirements traceability** in architecture DES.ar.5
interface design should be considered part of **software design**, or rather deserves its own, separate
process experts who do not feel expert in design, and **software design**ers who do not feel expert in
www.cs.vu.nl/~hans/publications/step2002/paper.pdf

Using Design Explanation within Formal Object Oriented - Method Lemai Nguyen (Correct)
to requirements are unpredictable, the **requirements traceability** and the rationale underlying the
collaborative work (Kaplan et al.1992) and **software design** (MacLean et al.1991 Lee, 1990) suggests
www.deakin.edu.au/mis/pages/staff/pswatman/pdf/J_Nguyen_SwatmanAndShanks_REJ1999.pdf

Requirements for the Use of COTS Operating Systems .. - Pierce, Wilson.. (Correct)
used to establish or validate all safety **requirements? Traceability** must be extended to the operating
from conceptual design and hazard analysis to **software design**. The second group of requirements considers
users.wmin.ac.uk/~beusdul/papers/dasia99.pdf

CREWS: Towards Systematic Usage of Scenarios, Use Cases and Scenes - Jarke (1999) (Correct)
Validation Based on Text Scenarios 4.2 **Requirements Traceability** and Change Envisionment Based on
iterative development, and manageable **software design** object models. Consistent with these
sunsite.informatik.rwth-aachen.de/pub/CREWS/CREWS-99-02.ps.gz

Reducing The Complexity Of The Software Design Process With.. - Schuler (2001) (Correct)
is executed by some part of the design, a **requirements traceability** matrix is constructed. The matrix
Reducing The Complexity Of The **Software Design** Process With Object-Oriented Design M. P.
techreports.larc.nasa.gov/pub/techreports/larc/91/conf-tech2001-v1.ps.Z

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

CiteSeer - citeseer.org - Terms of Service - Privacy Policy -

CiteSeer Find: matrix and functional requirements ar Documents Citations

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Order: citations weighted by year.**

Conceptual Modeling of MPEG-7 Description Schemes - Smith, Brigger, Li (1999) (Correct) (27 citations)
Programs Domain Logical Conceptual **Functional Requirements** Collection and Analysis Design Design
we discuss conceptual modeling as is used in **software design** and database design. By high-lighting the
to be used for conceptual design, class **design and software** implementation. 3 Conceptual Model of
www.ctr.columbia.edu/~ana/MPEG7/./download/M4775.doc.pdf

Consistency Checking of SCR-Style Requirements Specifications - Heitmeyer, Labaw, Kiskis (1995) (Correct) (49 citations)
more than a decade ago to describe the **functional requirements** of software unambiguously and concisely
and another system description (such as a **software design** or source code)the third class of formal
www.itd.nrl.navy.mil/ITD/5540/personnel/../publications/CHACS/1995/1995heitmeyer-isre.ps

From Non-Functional Requirements to Design through Patterns - Gross, Yu (2000) (Correct) (3 citations)
1 From Non-**Functional Requirements** to Design through Patterns Daniel
the relationships between nonfunctional requirements, **functional requirements**, and functional solution
[7,8] and to deal with change [9]In the **software design** area, the concept of design patterns has
www.cs.toronto.edu/~gross/rej/rejWithAuthorNames.pdf

On Formal Requirements Modeling Languages: RML Revisited - Greenspan, Mylopoulos.. (1994) (Correct) (16 citations)
aspect of software specifications are non-**functional requirements** such as efficiency, security and
functional specifications. The stage preceding **software design** has been frequently called specification,
dse.doc.ic.ac.uk/pub/requirements/greenspan.ps

From Single-User Architectural Design to PAC*: a Generic.. - Calvary, Coutaz, Nigay (1997) (Correct) (7 citations)
of the system, i.e.organising the **functional requirements** of the system into simpler conceptual
and make explicit the design steps that most **software design**ers in HCI tend to blend in a fuzzy way.
iihm.imag.fr/publs/1997/CHI97_PAC.ps.gz

CASL: The Common Algebraic Specification Language - Astesiano, Bidoit.. (2001) (Correct) (1 citation)
language for the formal specification of **functional requirements** and modular design of software. It has
of **functional requirements** and modular **software design**. Tools for Casl are interoperable, i.e.
www.dcs.ed.ac.uk/home/dts/pub/casl.ps

Tropos: A Framework for Requirements-Driven Software.. - John Mylopoulos Jaelson (2000) (Correct) (2 citations)
eventually lead to the functional and non-**functional requirements** of the system-to-be [6]In i* which
concepts to align requirements analysis with **software design** and implementation makes perfect sense. For
www.cs.toronto.edu/~mkolp/tropos/Fossil.pdf

High-Level Design and Architecture of an HTTP-Based.. - Neumann, Zdun (2000) (Correct) (2 citations)
application of xoComm, imposing several **functional requirements**, like support for local invocations,
to preserve good design ideas. Object-oriented **software design** patterns describe situations in which
nestroy.wi-inf.uni-essen.de/xotcl/xocomm.ps

Architecture of a Source Code Exploration Tool: A.. - Timothy Lethbridge.. (1997) (Correct) (6 citations)
. Detailed requirements about usability. **Functional requirements**. The system shall: F1 Provide search
writings can only be found in inaccessible **software design** documentation. To improve the state of
www.csi.uottawa.ca/~tcl/papers/Cascon/TR-97-07.pdf

Studying Software Architecture Through Design Spaces and Rules - Lane (1990) (Correct) (21 citations)
an appropriate system design based on **functional requirements**. The design space is useful in its own
work should be viewed as a means of codifying **software design** knowledge for use in day-to-day practice and
www.sei.cmu.edu/pub/documents/90.reports/ps/tr18.90.ps

Formal Specification of an Auctioning System Using.. - van den Berg.. (1999) (Correct) (2 citations)
are mainly characterised by their non{**functional requirements**, such as limited system resources or the
way when the team was formed to start the **software design** of this subsystem. There was no time to get
www.csr.ncl.ac.uk/vdm/vdmworkshop/./papers/vba/paper.ps

On the Role of Connectors in Modeling and Implementing.. - Oreizy, Rosenblum.. (1998) (Correct) (3 citations)
'style) and would then begin to allocate **functional requirements** to design elements. The design elements
the viewpoint of traditional approaches to **software design**, they are nevertheless problematic from a
www.ics.uci.edu/~peymano/papers/TR-UCI-ICS-98-04.pdf

Fuzzy Concepts and Formal Methods: A Fuzzy Logic Toolkit for Z - Matthews, Swatman (2000) (Correct) (1 citation)
to capture the elasticityof'soft' **functional requirements** has been proposed as a technique for
validation# to a more detailed and concrete **software design** document. Unlike some of the more informal
www.deakin.edu.au/mis/pages/staff/pswatman/pdf/C_MatthewsAndSwatman_ZB2000.pdf

Using COSMIC-FFP to Quantify Functional Reuse in Software.. - Ho, Abran, Oligny (2000) (Correct) (1 citation)
According to this model, software **functional requirements** are implemented by a set of functional
cycle, such as at the requirement analysis or **software design** stage. Software measurement can play an
www.lrgl.uqam.ca/publications/pdf/546.pdf

A Method for the Design and Development of Distributed.. - Born, Holz, Kath (2000) (Correct) (1 citation)
domain. These aspects span from **functional requirements** (e.g. transactionality) on object
to business process modeling, it is not a **software design** method in that sense, it heavily bases on
www.informatik.hu-berlin.de/~holz/Literatur/tools_pacific.pdf

A Pattern-Based Approach to Model Software Performance - Merseguer, Campos (2000) (Correct) (1 citation)
software engineer will be able to capture **functional requirements** as well as performance requirements. In
1. INTRODUCTION It is widely recognized that **software design** is a hard task that requires a significant
siui02.si.ehu.es/~jirgbdat/PUBLICATIONS/wosp00.ps.gz

Automatic Synthesis of Multi-Tasking Implementations from .. - Saksena, Karvelas, Wang (2000) (Correct) (1 citation)
design model that addresses the **functional requirements** of the system, and given end-to-end
the implementation choices. Thus, the **software design** models are separated into: 1) an
www.cs.pitt.edu/~manas/papers/isorc00.pdf

PAC-ing the Architecture of Your User Interface - Coutaz (1997) (Correct) (3 citations)
[29]Direct manipulation introduced new **functional requirements** on software architectures including the
models, such as PAC, are available for the **software design** of interactive systems. These design
ftp.imag.fr/imag/IIHM/ENGLISH/publications/1997/DSVIS97_PACing.ps.gz

The Methodology of N-Version Programming - Avizienis (1995) (Correct) (4 citations)
the initial specification is to state the **functional requirements** completely and unambiguously, while
on the principle of finding and eliminating **software design** faults either before or during the
www.informatik.tu-muenchen.de/~deiler/chap02.ps

First 20 documents Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

CiteSeer Find: functional requirements and software Documents Citations

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Only retrieving 125 documents (System busy - maximum reduced). Order: citations weighted by year.**

CASL: The Common Algebraic Specification Language - Astesiano, Bidoit.. (1999) (Correct) (1 citation)
language for the formal specification of **functional requirements** and modular design of software. It has
of **functional requirements** and modular **software design**, subsuming many previous specification
wwwat.mimuw.edu.pl/~tarlecki/kbn96grant/papers/AT2-vii99.ps

Performance Validation at Early Stages of Software Development - Smith, Woodside (1999) (Correct) (1 citation)
notion of what constitutes well-defined **functional requirements** they are often naive about performance
.Communicating state machines are captured in **software design** languages such as UML, SDL and ROOM, and in
www.perfeng.com/papers/smitwood.pdf

A Verification Tool Developer's Vade Mecum - Stevens (1999) (Correct) (1 citation)
a problem for verification tools, whose **functional requirements** are rather easy to define. Verification
may or may not appreciate the importance of **software design**: but even if they do, as humans they are not
www.dcs.ed.ac.uk/home/pxs/./vademecum.ps

A Security Model For Military Message Systems - Landwehr, Heitmeyer, McLean (1984) (Correct) (8 citations)
[8]available on the ARPA network. **Functional Requirements**. Message system operations may be
examples of a requirements document and a **software design** for such systems. In this paper, we
www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/Before1990/1984landwehr-tocs.ps

The Olympus Attitude and Orbital Control System - A.. - Burns, Wellings.. (1993) (Correct) (3 citations)
the system, giving the functional and non-**functional requirements**. The design of the system using the
and scheduled by a cyclic scheduler. 3. **Software Design** We represent the redesign of the AOCS using
www.cs.york.ac.uk/ftpdir/reports/YCS-93-190.ps.Z

Key Activities for a Development Methodology of.. - Bodart.. (1996) (Correct) (2 citations)
Context Analysis Ergonomic Requirements **Functional Requirements** User Interface Requirements Aaaa Aaaa
Design Context Analysis Ergonomic Requirements **Functional Requirements** User Interface
(fig. 1)Presentation Design Dialogue **Design Software** Architecture Design Context Analysis
ftp.info.fundp.ac.be/pub/publications/RP/RP-96-025.ps.Z

Performance Analysis of Communication Systems Formally Specified .. - Steppler (1998) (Correct) (1 citation)
3 language. There are functional and non-**functional requirements**, e. g. expected performance, security,
first presents the SDL-oriented method of **software design** and then points out how to handle the
www.dfv.rwth-aachen.de/~steppler/publications/steppler_wosp98.ps.gz

Mapping an Embedded Hard Real-Time Systems SDL Specification.. - Kolloch, Färber (1998) (Correct) (1 citation)
both the expression of functional and non-**functional requirements** and supports the verification of these
usage of a formal specification methodology in **software design** will reduce the development effort,
ftp.lpr.e-technik.tu-muenchen.de/pub/papers/rtsg/lctes98-kolloch.ps.gz

The Field of Software Architecture - Hofmann, Horn, Keller, Renzel.. (1996) (Correct) (1 citation)
that is relevant to the software system. **Functional requirements** determined by the application domain are
task. Complex systems have to fulfil many **requirements, functional** and nonfunctional ones. A well-organized
the project `Bellevue' 2 TU Mnchen 3 **software design** &management (sd&m) GmbH &Co. KG, Mnchen
www4.informatik.tu-muenchen.de/reports/TUM-I9641.ps.gz

Software Engineering - Sempa Software Engineering (1987) (Correct) (2 citations)
environment. ParTfC has no specific **functional requirements** -except that it must compute the same
of this cycle, see the sidebar, The **software-design** process in parallel scientific computing.
wwwbode.in.tum.de/archiv/artikel/ieee-concurrency/sempa.pdf.gz

An Empirical Study of Architectural Design Operations - Kazman, Reddy (Correct) (2 citations)
empirical software engineering, non-**functional requirements** 1 Introduction The purpose of this
that have been derived through a study of the **software design** literature and interviews with expert

pukapuka.inrialpes.fr/Olan/Docs/Bibliotheq[illegible]SE_SA/KR96.ps.gz

An Engineering Approach to Hard Real-Time System Design - Kopetz, Zainlinger... (1991) (Correct) (1 citation)
the fact that the designer has, apart from **functional requirements**, to face fault tolerance aspects as
the underlying system architecture. However, **software design** and test methodology must carefully
www.vmars.tuwien.ac.at/papers/ps-files/rr-01-91.ps.gz

Scenario-based Assessment of Software Architecture Usability - Folmer, van Gurp, Bosch (Correct)
of the software architecture based on the **functional requirements**. Although software engineers will not
.It is required to determine when the **software design** process is finished. Therefore, assessment
www.eelke.com/research/icseworkshop.pdf

Software---Practice And Experience, Vol. 23(3), 293--315., - Visualizing Hierarchy Of (Correct)
finer granularity as more performance and **functional requirements** become available. This process involves
for real-time applications. Modeling both **software design** and hardware resources is essential for a
www.cs.ubc.ca/local/reading/proceedings/spe91-95/spe/./vol23/issue3/spe812.pdf

Linking Patterns and Non-Functional Requirements * - Ivdm Araujo And (Correct)
Linking Patterns and Non-**Functional Requirements** *Ivdm Araujo and Michael Weiss School
with the adaptation of the pattern approach to **software design** and given the yearly deliberations to
jerry.cs.uiuc.edu/~plop/plop2002/final/PatNFR.pdf

Fuzzy Concepts and Formal Methods: Some Illustrative Examples - Matthews (1999) (Correct)
heuristically. The concept of a soft **functional requirement** -one where the degree to which the
validation) to a more detailed and concrete **software design** document. Unlike some of the more informal
www.deakin.edu.au/mis/pages/staff/pswatman/pdf/C_MatthewsAndSwatman_APSEC2000.pdf

Requirements Engineering in Agile Software Development - Paetsch (2003) (Correct)
. 42 4.11 Non-**functional requirements** .
model are: requirements definition, system and **software design**, implementation and unit testing,
sern.cpsc.ucalgary.ca/~milos/papers/2003/DA_Frauke.pdf

Fuzzy Concepts and Formal Methods - Matthews, Swatman (Correct)
heuristically. The concept of a soft **functional requirement**-onewherethedegreetowhich the
validation) to a more detailed and concrete **software design** document. Unlike some of the more informal
www.deakin.edu.au/mis/pages/staff/pswatman/pdf/BC_MatthewsAndSwatman2002.pdf

Designing Web-Based Systems in Social Context: - Goal And Scenario (Correct)
achieve, which, generally, describe the **functional requirements** of the target information system. In GRL
It involves domainspecific knowledge, generic **software design** knowledge and knowledge about the specific
www.mm.di.uoa.gr/~rouvas/ssi/caise2002/23480037.pdf

Use of UML for modeling non-functional aspects - Guadalupe Salazar-Zrate Pere (Correct)
software components .Keywords :UML, non-**functional requirements**, component **software design**. 1 M.G.
: UML, non-**functional requirements**, component **software design**. 1 M.G. Salazar-Zrate is holder of a
www.lsi.upc.es/~gessi/comprolab/psfiles/paperICSSEA.pdf

Documents 21 to 40 Previous 20 Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

CiteSeer - citeseer.org - Terms of Service - Privacy Policy - Copyright © 1997-2002 NEC Research Institute

CiteSeer Find: functional requirements and software Documents Citations

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Order: citations weighted by year.**

Use of UML for modeling non-functional aspects - Guadalupe Salazar-Zrate Pere (Correct)
software components .Keywords :UML, non-**functional requirements**, component **software design**. 1 M.G.
: UML, non-**functional requirements**, component **software design**. 1 M.G. Salazar-Zrate is holder of a
www.lsi.upc.es/~gessi/comprolab/psfiles/paperICSSEA.pdf

Quantitative Aspects of Requirements Evolution - Anderson, Felici (Correct)
Requirements Maturity Index. The software **functional requirements** fall into 8 functions for which separate
flowed down to the software and hardware **requirements. Functional** and Operational Requirements. The
task can be split into two broad areas: **software design** and code and verification. Two main
www.dcs.ed.ac.uk/home/mas/doc/cameraready_compsac2002.pdf

Aspects + GAMMA = AspectGAMMA: A Formal Framework.. - Mousavi.. (Correct)
etc.should be kept independent from **functional requirements**, as well as others so that change and
is one of the concepts at the core of modern **software design** and evolution. It has been advocated as a
www.win.tue.nl/~michelr/techreports-michelr/UCI-CECS-02-01.pdf

From specifications to code in Casl - Aspinall, Sannella (2002) (Correct)
intended for formal specification of **functional requirements** and modular **software design** and
of **functional requirements** and modular **software design** and subsuming many previous specification
www.dcs.ed.ac.uk/home/dts/pub/amast2002.ps

Refinement and Verification Applied to an.. - Fokkink.. (Correct)
of the AIDA system, starting from the **functional requirements**. Keywords: refinement, verification,
modelling, software interface specification, **software design**, implementation and maintenance. It targets
www.cwi.nl/~wan/WJFmore/ktvfm.ps

Tolerating Architectural Mismatches - de Lemos, Gacek, Romanovsky (2002) (Correct)
have to meet many functional and non-**functional requirements** which often have to be expressed at the
static methods of correcting mismatches (c.f. **software design** faults)First of all, complex applications
www.cs.ukc.ac.uk/events/conf/2002/wads/Proceedings/delemos.pdf

Client/Server Architectures for Business Information Systems: .. - Renzel, Keller (1997) (Correct)
so that my users' functional and non-**functional requirements** are met? There are several answers to
requirements. It significantly influences the **software design** and requires a very careful analysis of the
www.wellm.de/arcus/publicat/cs_plop97.ps.gz

Unknown - (Correct)
software architectures in the context of **functional requirements** (a.k.a. component-based designs)to
Presented at Workshop on New Visions for **Software Design** and [Productivity: Research and
performance)It is possible to automatically **design software** (i.e.produce an equation that optimizes
www.cs.utexas.edu/users/schwartz/ExCIS.pdf

Software Product Measurement For Supplier Evaluation - Alain April Is (Correct)
are further categorized in 7 areas: 1) **Functional Requirements** 2) Performance Requirements 3)
At this step the team will review the **software design** and coding guidelines of the supplier, if
www.lrgl.uqam.ca/publications/pdf/583.pdf

Report Series - National Institute Of (Correct)
components of the VTS software, but two **functional requirements** are of particular importance: Schema
The audience for this document includes **software design**ers and developers. This document builds on
www.nist.gov/msidlibrary/doc/morris91.ps

Suggested Software Engineering Practices. - Alma Phase Harris (1999) (Correct)
actors Identify the functional and non-**functional requirements** Identify risks 2.2.1.2 Artifacts
Overall system description Software **requirements -functional** and non-functional Actors Risk
7 3.2 SDD -The **Software Design** Document

granada.iram.es/FutureControl30M/wg_sw_eng/SEpractices.pdf

Towards a Problem-Driven Approach to Perspective-Based.. - Chen, Poon, Tang, Tse, Yu (2002) (Correct)
because the document should define all the **functional requirements** and constraints. As a result, reviewers
Since these documents form the basis for **software design** and implementation, they should be complete,
www.csis.hku.hk/research/techreps/document/TR-2002-01.pdf

Reading Techniques for OO Design Inspections - Travassos, Shull, Carver, Basili (2002) (Correct)
there will be a textual description of the **functional requirements** that may also describe certain behaviors
early in software development. Inspections of **software design** can be especially crucial since design
www.cs.umd.edu/Library/TRs/CS-TR-4353/CS-TR-4353.pdf

Analysing Software Requirements Specifications for Performance - Dorin Petriu Murray (Correct)
Use Case Maps, completion, non-**functional requirements** 1 Introduction The earliest decisions
the context of automated model-building from **software design** products, which may be part of the
www.sce.carleton.ca/ftp/pub/cmw/petriu-req.pdf

Using UML to Reflect Non-Functional Requirements - Cysneiros, Leite (2001) (Correct)
Using UML to Reflect Non-**Functional Requirements** Luiz Marcio Cysneiros Department of
by the system, and therefore affects the **software design**. NFRs such as "small learning curve" or
www.cs.toronto.edu/~cysneiro/articles/Cascon.pdf

Unifying User-Centered and Use-Case Driven.. - Antunes Seffah.. (2001) (Correct)
or tasks flowcharts to supplement the **functional requirements** analysis. However, for large-scale
iterative development and manageable **software design** object models. Ralyte (1999) in the CREWS
www.hci.uu.se/~jg/UCD2001/Antunes2.pdf

Conflicts and Trade-offs between Software.. - Lundberg.. (Correct)
for 10 weeks. The groups had identical **functional requirements**. One group (DMO 1) was allowed to use a
its performance requirements using a certain **software design** on a certain platform. An equally
www.ide.hk-r.se/~dha/chapter1.PDF

A Family of Reading Techniques for OO Design Inspections - Guilherme Travassos Forrest (2000) (Correct)
activities are concerned with taking the **functional requirements** and mapping them to a new notation or
early in software development. Inspections of **software design** may be especially crucial since design
www.cs.umd.edu/projects/SoftEng/ESEG/papers/pdf/wqs00.pdf

Validation of Dynamic Behavior in UML Using Colored Petri Nets - Pettit, IV, Gomaa (Correct)
a use case model is developed in which the **functional requirements** of the system are defined in terms of
of concurrent and real-time object-oriented **software designs**. 1 Introduction This paper presents an
www.disi.unige.it/person/ReggioG/UMLWORKSHOP/Pettit.pdf

Documents 41 to 60 Previous 20 Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

CiteSeer Find: functional requirements and software | Documents | Citations

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Order: citations weighted by year.**

Component Indicators for Architecture Interaction Assessment - Payton, Davis, Gamble.. (Correct)
properties, protocol information, and non-**functional requirements**. Together they form an architecture
of developing a system that fulfills its **requirements, functional** or otherwise. 2.2 Interface Definition
conflicts during system 2 **design. Software** architecture offers a view of the design in
www.seat.utulsa.edu/publications/PDGF00.pdf

Quality Management Applied to Undergraduate Software.. - Aj Walker Software (1994) (Correct)
is managed by identifying and elaborating **functional requirements**, and tracking these through the software
One of these, entitled Towards defect-free **software design** provided the catalyst for a research
www.seal.ac.za/1994_01/qs16331.pdf

Streams, Structures, Spaces, Scenarios, Societies.. - Gonçalves.. (Correct)
of a computation in order to accomplish a **functional requirement**. Societies comprehend entities and the
that do not benefit from digital library and **software design** experience. Wasted e#ort and poor
csgrad.cs.vt.edu/~mgoncalv/5s5.pdf

Schemebuilder, A Design Aid For The Conceptual.. - Bracewell.. (1993) (Correct)
structured by decomposing the high level **functional requirements** specification into a nested hierarchy of
for dynamic simulation supporting embedded **software design** and development monitoring system
www-edc.eng.cam.ac.uk/~rhb24/iced93.pdf

Designing Real-Time Applications with the COMET/UML Method - Gomaa (Correct)
a use case model is developed in which the **functional requirements** of the system are defined in terms of
it is necessary to synthesize an initial **software design** from the analysis carried out so far. In the
wooddes.intranet.gr/papers/gomaa.pdf

Requirements Evolution From Process to Product Oriented.. - Anderson, Felici (2001) (Correct)
the RMI calculated for all the software **functional requirements**. In this case the RMI results to be
flowed down to the software and hardware **requirements. Functional** and Operational Requirements. The
can be split into two broad areas, that of **software design** and code and the other of verification. Two
www.dcs.ed.ac.uk/home/mas/doc/profes2001.pdf

Integrated Process Control and Data Management in.. - Welsh, Kalathil.. (Correct)
Detailed Design Support Workflows System **Requirements Functional** Design Chassis Design Reusable Design
design processes. Development of systems and **software design** processes, as well as enhanced supporting
www.pidworld.com/_hdl/1/VHDL_Internet/papers/welsh.pdf

The Common Framework Initiative for algebraic specification and.. - Sannella (1999) (Correct)
called Casl for formal specification of **functional requirements** and modular **software design** which
of **functional requirements** and modular **software design** which subsumes many previous algebraic
www.dcs.ed.ac.uk/home/dts/pub/wadt2001.ps

Fds - A Functional Design Software System - Kirschman, Fadel (Correct)
design is decomposed into a hierarchy of **functional requirements** (FRs) which map directly to design
0 Fds -A Functional **Design Software** System C.f. Kirschman And G.m. Fadel Center
1 Fds -A Functional **Design Software** System C.f. Kirschman And G.m. Fadel Center
www.vr.clemson.edu/credo/papers/postscript/FDS.pdf

OSPF Efficient LSA Refreshment Function in SDL - Monkewich, Sales, Probert (2001) (Correct)
UCM diagrams were used to capture key **functional requirements** as sequences of actions and reactions
5 selected business values and compared to the **software design** approaches based on manual programming
www.sce.carleton.ca/ftp/pub/UseCaseMaps/sdl01-sales.pdf

Designing Electronic Voting - Murk (2001) (Correct)
.15 1.6.1 **Functional Requirements** .15
must at least satisfy the following requirements: **Functional Requirements** System must allow forming

that design for such system does not mean only **software design**, but also design of the organization and the
www.cs.ut.ee/~olegm/papers/evdes.ps.gz

Computing from Unconventional Viewpoints - Intellectual Universe Of (Correct)
the application domain are necessary since **functional requirements** are expressed using technical terms of
see it. b) Functional &Non-**Functional Requirements**: **Functional requirements** include the concepts and
Engineering p Requirements Engineering p **Software Design** p ffl Practice: Systems Engineering
www.it.dtu.dk/~db/kansai/paper.ps

Supporting Extension of Components with new Paradigms - Dominick, Ostermann (2000) (Correct)
concerns, which are independent of the **functional requirements** of the application, have proved
Traditional solutions in this area rely on **software design** patterns and have been successfully employed
trese.cs.utwente.nl/Workshops/OOPSLA2000/papers/dominick.pdf

Program Reliability Estimation Tool - Reza Nejat Software (Correct)
that the software satisfies its specified **requirements (Functional** testing, 19]Structural testing
to confirm the correct implementation of the **software design** (Structural testing [3]and 2) to confirm
www.crl.mcmaster.ca/SERG/papers/391.ps.Z

Formal Methods in the 21'st Century: An Assessment of Today -.. - Bjørner (1998) (Correct)
Requirements: In order to express **functional requirements** we need clarify the domain. ffl Domain
tools. 3 Domain /Requirements /Software **Software Design**: In order to **design software** we need
/ Software **Software Design**: In order to **design software** we need requirements. Requirements: In order
www.it.dtu.dk/~db/japan/icse98/icse98.ps

Structure and Style in Use Cases for User Interface Design - Constantine, al. (2000) (Correct)
design of communicating objects to satisfy **functional requirements**. Success in all these endeavors rests on
and user interface design on the one hand and **software design** and development on the other, part of the
and their consequences for user interface **design and software** usability. Common narrative styles are
www.foruse.com/Files/Papers/structurestyle2.pdf

Assessing Optimal Software Architecture Maintainability - Bosch, Bengtsson (Correct)
of the software architecture based on the **functional requirements** specified in the requirement
requirements. Based on these relations, **software design** is often performed as an implicit,
www.cs.rug.nl/~bosch/papers/OptimalSAMaintainabilityCSMR01.pdf

Evaluated Object-Oriented Software Development - Dumke, Foltin (Correct)
-constraints, given situation, **functional requirements**, management requirements
complexity, by Arora et al for the real-time **software design** in CArora et al 95/and by Lorenz as
irb.cs.tu-berlin.de/~zuse/gi/dufol95.ps.gz

Practical Application of Functional and Relational.. - Lawford, McDougall, .. (Correct)
while still permitting the use of **functional requirements** speci cation and designs descriptions.
the properties of the design described in the **Software Design** Description (SDD)comparing them against
ftp.cs.uiowa.edu/pub/rus/AMASTpapers/p44.ps

Controlling Requirements Evolution An Avionics Case Study - Anderson, Felici (2000) (Correct)
(i.e.system requirements, software **functional requirements**, etc.The System Requirements cover
Requirements Elicitation System Process **Software Design** Testing Review Coding System Requirements
www.dcs.ed.ac.uk/home/mas/doc/af_safecomp2000.pdf

Documents 61 to 80 Previous 20 Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

CiteSeer - citeseer.org - Terms of Service - Privacy Policy -

CiteSeer Find: functional requirements and software Documents Citations

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Only retrieving 125 documents (System busy - maximum reduced). Order: citations weighted by year.**

Model Checking Verification and Validation at JPL.. - Schneider.. (1998) (Correct)
be exhaustively searched allowing critical **functional requirements** to be validated down to the design
used to model a complex spacecraft controller. **Software design** and implementation validation were carried
sel.gsfc.nasa.gov/website/sew/1998/topics/Schneider-p.pdf

Fuzzy Z? - Matthews, al. (1997) (Correct)
to capture the elasticity of `soft' **functional requirements** has been proposed as a technique for
validation) to a more detailed and concrete **software design** document. Unlike some of the more informal
ironbark.bendigo.latrobe.edu.au/staff/chrism/sydney2.ps

Fuzzy Concepts and Formal Methods: A Fuzzy Logic Toolkit for Z - Matthews, Swatman (2000) (Correct)
to capture the elasticity of `soft' **functional requirements** has been proposed as a technique for
validation) to a more detailed and concrete **software design** document. Unlike some of the more informal
ironbark.bendigo.latrobe.edu.au/staff/chrism/yorked2.ps

Fuzzy Concepts and Formal Methods: Some Illustrative Examples - Matthews, al. (1999) (Correct)
heuristically. The concept of a soft **functional requirement** -one where the degree to which the
validation) to a more detailed and concrete **software design** document. Unlike some of the more informal
ironbark.bendigo.latrobe.edu.au/staff/chrism/6479.2492_matthews.ps

Architecture-Based Software Engineering - Stafford, Wolf (1999) (Correct)
obvious the architecture must satisfy the **functional requirements** for the system. The extra-functional
a software system's architecture. To make the **software design** task tractable, skilled software architects
ftp.cs.colorado.edu/users/alw/doc/papers/CU-CS-891-99.ps.gz

Building a Funky Interface to a Web Search Engine - Study Submitted In (2000) (Correct)
well as the limitations of the system. The **functional requirements** of the prototype are illustrated and an
technique. The project also reports on the **software design** issues of the proposed web search engine.
dis.shef.ac.uk/mark/cv/publications/dissertations/Ye2000.pdf

Debugging VHDL Designs using Model-Based Reasoning - Wotawa (2000) (Correct)
others. The specification describes the **functional requirements** of a digital circuit and can be 3
design process becomes very similar to the **software design** process. Moreover, searching for faults in a
locating and fixing faults within a hardware **design or software** are rarely available. In this paper we
www.dbai.tuwien.ac.at/staff/wotawa/aieng00.ps.gz

Component-Based Groupware Tailorability using Monitoring.. - de Farias, Diakov (2000) (Correct)
from the application. Depending on the **functional requirements**, adding a new software extension to the
of new solutions for component-based **software design** and implementation. Component-based
amidst.ctit.utwente.nl/publications/cscw_cbg2000.pdf

An Object-Oriented Approach To The Co-Design Of.. - Machado, Fernandes.. (Correct)
are considered, with their additional non-**functional requirements** that enormously constrict the allowable
design (hardware engineers)for FMOTSs **design (software** engineers) and for final solutions design
shiva.di.uminho.pt/~miguel/PUBLI/mesc00.pdf

Active Object Oriented Databases in Control Applications - Loborg, Risch, Sköld, Törne (1993) (Correct)
subassemblies are placed on a pallet. The **functional requirements** on the application is that the assembly
regard to control by software. Typical for the **software design** problem in control applications are the
ftp.ida.liu.se/pub/labs/edslab/reports/LiTH-IDA-R-93-28.ps.gz

Client/Server Architectures for Business Information Systems - A.. - Renzel (Correct)
so that my users' functional and non-**functional requirements** are met? There are several answers to
requirements. It significantly influences the **software design** and requires a very careful analysis of the
st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/renzel.pdf

The Waveform Correlation Event Detection System Global.. - Judy Beiriger Susan (1997) (Correct)
of the WCEDS prototype, discussing **functional requirements**, operation, and information flow.
Event Detection System Global Prototype **Software Design** Judy I. Beiriger, Susan G. Moore, Julian R.
infoserve.sandia.gov/sand_doc/1997/973179.pdf

Unknown - It Is Common (Correct)
should be fed by the functional and non-**functional requirements** and by available design patterns
software does not meet these goals. sd&m **software design** &management GmbH Co. KG has been founded
www4.informatik.tu-muenchen.de/proj/arcus/TUM-I9746/Preface.ps.gz

DECNIS Architecture (digital document NIPG-007-01, December 1993) - Bryant (1993) (Correct)
currently available, we split the **functional requirements** into two sets: those best handled in a
The paper then identifies the key hardware and **software design** features, and finally provides a summary of
gatekeeper.dec.com/pub/DEC/DECNIS/whitepapers/DECNIS_architecture.ps

Architectural Blueprints - The "4+1" View Model of Software.. - Kruchten (1995) (Correct)
handle separately the functional and non **functional requirements**. Each of the five views is described,
The rest of the story is in the realm of **software design**, where, by the way, development may continue
software architecture, view, object-oriented **design, software** development process Introduction We all
plg.uwaterloo.ca/~migod/746/papers/kruchten-4plus1.pdf

Monitoring Extensions for Component-Based Distributed.. - Diakov, van Sinderen.. (2000) (Correct)
etc. Nevertheless, depending on the **functional requirements**, a new software extension to a legacy
of new solutions for component-based **software design** and implementation. There are numerous
amidst.ctit.utwente.nl/publications/proms2000.pdf

. System Design - There Are Two (Correct)
He must do so according to a number of **functional requirements**: the kitchen should be close to the
Design There are two ways of constructing a **software design**: One way is to make it so simple that there
wwwbruegge.in.tum.de/teaching/ss99/CBSE/book/SystemDesign042699.pdf

Manifest 3D: Framework Develop 3D Graphics Applications - Alfredo Teyseyre Ricardo (Correct)
(Coarse Fine-Grained Patterns) **Functional Requirements** Non-**Functional Requirements**
names and describes a common problem found in **software design** and prescribes adequate solution that
www.exa.unicen.edu.ar/~teyseyre/./papers/manifest3d-98.ps.gz

Reducing Uncertainty About Common-Mode Failures - Voas, Ghosh, Charron, Kassab (1997) (Correct)
that different programs have different **functional requirements**. For example, one program might do a
been adequately addressed. The staff considers **software design** errors to be credible common-mode failures
chacs.nrl.navy.mil/publications/CHACS/1997/1997kassab-ISSRE97.ps

Computer Science Environments for Group-Oriented Software.. - Emmerich, Schäfer (1996) (Correct)
and concurrently. Section 4 derives **functional requirements** of a support environment from language
No. 1996#02 Environments for Group-Oriented **Software Design** -The Groupie Experience Wolfgang Emmerich
www.cs.ucl.ac.uk/staff/W.Emmerich/publications/ASE/ase-submission.pdf

Documents 81 to 100 Previous 20 Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

Find: functional requirements and software Documents Citations

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Only retrieving 125 documents (System busy - maximum reduced). Order: citations weighted by year.**

An Agenda for Specifying Software Components with Complex.. - Winter, Santen, Heisel (1998) (Correct)
it is often possible to express the **functional requirements** as a direct relation between the values
for safety analyses, test case generation, and **software design**. We use the ESPRESS notation SZ [1] to
www.first.gmd.de/~kirsten/publications/sc98.ps.gz

Where do Software Architectures come from? - Systematic.. - Bjørner (1998) (Correct)
domain and of functional and non-**functional requirements**. A major aim &objective of this paper
requirements. 2.4.1 **Functional Requirements Functional requirements** "derive" from, or as
In this paper we show how details of a **software design** emerges in two steps: software architecture
www.it.dtu.dk/~db/jssst/paper.ps

Using Design Patterns - An observational study - Jacobsen (1999) (Correct)
including the functional and the non-**functional requirements** as well as the expectations to the
Requirements Logical Platform **Requirements Functional** Design Phase Developer Architecture
design process. Architecture One aspect of **software design** is an overall understanding of a specific
www.cit.dk/COT/reports/reports/Case6/15/cot-6-15.pdf

Client/Server - Study Software Engineering (Correct)
alternatives could have met the system non-**functional requirements**. The final learning objectivewas to
maintenance and implementation issues in **software design**. The case study, developed as a classroom
www.cs.cmu.edu/afs/cs.cmu.edu/project/vit/ftp/pdf/client_server.pdf

Surf: Achieving Quality Through Software Reuse - A Process.. - Riva, al. (Correct)
aspects such as satisfaction of the user's **functional requirements**, as well as execution speed, ergonomic
a team is generally composed only partially of **software design**ers. The head count specific for software
is.lse.ac.uk/helsinki/riva.pdf

Finding Mode Invariants in SCR Specifications - Jin (1995) (Correct)
more than a decade ago to describe the **functional requirements** of software, the SCR specifications has
and test cases that are independent of the **software design** structure. 1 Introduction A software
isse.gmu.edu/techrep/1995/95_112_offutt.ps

Reading Techniques for OO Design Inspections - Travassos, Shull, Carver, Basili (1999) (Correct)
activities are concerned with taking the **functional requirements** and mapping them to a new notation or
early in software development. Inspections of **software design** may be especially crucial since design
www.cs.umd.edu/projects/SoftEng/ESEG/papers/postscript/sew99.ps

Tools for Design Rationale Documentation in the Development of.. - University (Correct)
in addition of many functional and non-**functional requirements**. However, there are always much more
refine, organise and reuse knowledge for **software design**. Design Decision Tree is a partial
www.bell-labs.com/user/dep/prof/wicsa1/final/savolainen.pdf

The DECNIS 500/600 Multiprotocol Bridge/Router and Gateway - Bryant, Brash (1993) (Correct)
currently available, we split the **functional requirements** into two sets: those best handled in a
The paper then details the hardware and **software design** and concludes with a performance summary.
www.europe.digital.com/info/DTJ907/DTJ907PF.PDF

Using Non-Functional Requirements in Component-Based.. - Botella, Franch, Burgués (Correct)
Using Non-**Functional Requirements** in Component-Based Software
Key words and phrases: component **software design**, non-**functional requirements**, prototyping. 1
www.lie.us.es/pub/isi/jids/botella@morfeo.upc.es.ps.gz

Formal Development of A Toll Way Control System - Indrika (1995) (Correct)
. 3 2.2.1 **Functional Requirements** .
modelling, 2) requirements capture, and (3) **software design** &programming are being investigated, c)

ftp.iist.unu.edu/pub/techreports/report45.ps.gz

Analysing Socio-Technical System Requirements - Sutcliffe, Minocha (1999) (Correct)
analyses between conflicting goals and non-**functional requirements**, but it does not provide an analytic analysis derived from concepts in modular **software design** (DeMarco 1978) and organisational theory 14915) ISO 14915-1 Multimedia User Interface **Design -Software** Ergonomic Requirements -Part 1:
sunsite.informatik.rwth-aachen.de/pub/CREWS/CREWS-98-37.ps.gz

The Common Framework Initiative for algebraic specification and.. - Sannella (1999) (Correct)
called Casl for formal specification of **functional requirements** and modular **software design** which of **functional requirements** and modular **software design** which subsumes many previous algebraic
www.dcs.ed.ac.uk/home/dts/pub/psi.ps

A Control Architecture to Achieve Manipulation Task.. - Cho, Park, Park, Oh, Lee (Correct)
In this paper, we first consider the **functional requirements** for the control architecture of humanoid learning capability should be considered to **design software** architecture of humanoid robots. 3. A
amadeus.kist.re.kr/members/cyj/icra98.ps

DSSA-ADAGE Operational Scenarios and System Vision - Tracz, Coglianese (1992) (Correct)
scenario is a means of specifying the **functional requirements** of a system. An operational scenario is -an annotated record of the system and **software design** decisions and rationale resulting from the
www.owego.com/dssa/lm-docs/IBM9201.ps

I. Reusable Software Catalogues - Design and Retrieval Support - Weber, Casais (Correct)
searches for interfaces that fulfill his **functional requirements**. He may do this by learning the concepts class hierarchy. As in classical modular **software design**, interfaces must be separated from their
ftp.fzi.de/pub/PROST/papers/catalogues.ps.Z

Literature Survey on User Requirements Specification and .. - (Some), Dssouli, Vaucher (Correct)
B. 1992)Representing and Using Non-**Functional Requirements**: A Process-Oriented Approach. IEEE acquisition methods and the two type of **requirements: functional** require6 expansive to correct (Boehm, M. E.McGowan, C.and Ross, D. T. 1978)**Software design** using SADT. Structured Analysis and Design,
ftp.crim.ca/igloo/privee/livrables/M.c.1-SSo-requirements-survey-v1.0.ps.Z

Applying Case-Based Reasoning to Software Quality Management - Lees, Hamza (Correct)
company requirements, known as design or **functional requirement** these are generally global product compared to the functional decomposition of a **software design**. An overall quality measure for the type, function, technology used, method of **design, software** specifications, the importance of software
www.informatik.hu-berlin.de/~cbr-ws/GWCBR96/PAPERS/leesetal.ps.gz

Client/Server Distribution - A Pattern Language - Renzel, Keller (1997) (Correct)
so that my users' functional and non-**functional requirements** are met? 1 This work has been published requirements. It significantly influences the **software design** and requires a very careful analysis of the
www4.informatik.tu-muenchen.de/proj/arcus/TUM-I9746/2.4.ps.gz

The Role of HCI in CASE Tools Supporting Formal Methods - Connie Heitmeyer (1994) (Correct)
been proposed for formally representing **functional requirements**, the SCR tabular approach is one of the quality of thetoolset' user interface and its **software design** [4]A high-quality user interface would
www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1994/1994heitmeyer-ICSE.ps

Documents 101 to 120 Previous 20 Next 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

CiteSeer - citeseer.org - Terms of Service - Privacy Policy - Copyright © 1997-2002 NEC Research Institute

CiteSeer Find: functional requirements and software [Documents] [Citations]

Searching for **functional requirements and software design**.
Restrict to: Header Title Order by: Citations Hubs Usage Date Try: Amazon B&N Google (RI) Google (Web) CSB DBLP
127 documents found. **Order: citations weighted by year.**

The Role of HCI in CASE Tools Supporting Formal Methods - Connie Heitmeyer (1994) (Correct)
been proposed for formally representing **functional requirements**, the SCR tabular approach is one of the quality of thetoolset' user interface and its **software design** [4]A high-quality user interface would
www.itd.nrl.navy.mil/ITD/5540/publications/CHACS/1994/1994heitmeyer-ICSE.ps

The Design of Hybrid Systems using the SEA Environment - Rust, Stroop, Tacken (1997) (Correct)
of systems under consideration, not only **functional requirements** of individual system parts must be where the main focus is the view point of **software design** for embedded systems. Due to the hybrid
www.c-lab.de/~aatools/publications/papers/1997/cr97_20.ps.gz

Bell Communications Research 445 South Street, Morristown, NJ .. - Cameron Kcheng (Correct)
as refinement of the service description **requirements, functional** analysis, network requirements transforms a high-level design into a detailed **software design** and develop the necessary software
thumper.bellcore.com/pub/fjlin/fiwpos97.ps

A Model-Based Tool for Finding Faults in Hardware Designs - Stumptner, Wotawa (1996) (Correct)
with a specification delineating the **functional requirements**. Figure 2: A typical waveform trace design is increasingly getting similar to the **software design** process, and the search for faults in the
www.dbai.tuwien.ac.at/staff/wotawa/aid96.ps.gz

Technology Transfer: Software Engineering and Engineering.. - Finkelstein, Nuseibeh (Correct)
the specification of socalled "non-**functional requirements**" that deal with aspects of systems that software development and instrument **design. Software** Engineering. While computer scientists
dse.doc.ic.ac.uk/dse-papers/seed/cc92.ps.Z

Software Architecture for Collaborative Development: A.. - Abbas, Kazmierczak, Dart (1998) (Correct)
behavioural aspects: functional and non-**functional requirements** and properties of the overall system and
Keywords Software architecture, **software design**, software architecture properties,
Keywords Software architecture, **software design**, software architecture properties, collaborative
munkora.cs.mu.oz.au/publications/tr_db/./mu_98_14.ps.gz

A New Programming Paradigm for Engineering Design Software - Salustri, Venter (1994) (Correct)
design information (including aspects of **functional requirements** and design intent) effectively and
A New Programming Paradigm for Engineering **Design Software** F. A. Salustri R. D. Venter Department of
"A New Programming Paradigm for Engineering **Design Software**,Engineering with Computers, 10, pp.
salustri.esxf.uwindsor.ca/~fil/Papers/designer-ewc94.ps

Documents 121 to 127 Previous 20

Try your query at: Amazon Barnes & Noble Google (RI) Google (Web) CSB DBLP

79545

U.S. DEPARTMENT OF COMMERCE
Patent and Trademark Office

DOCUMENT RETRIEVAL REQUEST FORM

Requester's Name: TUAN VU	Case Serial Number: 09731678	Art Unit/Org.: 2124
Phone: 305 7207	Fax:	Building: PK2
Room Number: 5Y18		

Class/Sub-Class: 717

Date of Request: 9-4-03 Date Needed By:

Paste or add text of citation or bibliography: Paste Citation Only one request per form. Original copy only. □

Author/Editor:	S.J. KIM, N.P. SUH, S.K. KIM
Journal/Book Title:	Annals of the CIRP
Article Title:	Design of software systems based on axiomatic design
Volume Number:	40 Report Number: 1 Pages: 165-170
Issue Number:	Series Number: Year of Publication: 1991
Publisher:	
Remarks: 78 NO CITATION	please call me if this require more than 1 day. 462684

Staff Use Only

Monthly Accession Number:

Library Action	PTO		LC		NAL		NIH		NLM		NIST		Other	
	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd	1st	2nd
Local Attempts	✓													
Date	9/5													
Initials	Sq													
Results	N/A													
Examiner Called														
Page Count														
Money Spent														

		Source	Date
Remarks/Comments 1st and 2nd denotes time taken to a library O/N – Under NLM means Overnight	Ordered From:	CISTI ordered Complete, faxed	
	Comments:	Please get copy of CIRP version AND a copy of the attached citations article Thx Sq	

PTO-XXXX (2-96) USCOM-DC

09/05/03 11:02 ☎7033065509 USPTO EIC 001/002

Design of Software System Based on Axiomatic Design

S.-J. Kim, N. P. Suh (1), Massachusetts Institute of Technology, Cambridge, MA/USA;
S.-G. Kim, Korea Institute of Science and Technology, Seoul/Korea

Received on January 15, 1991

Abstract

The ability to utilize the fully automated flexible manufacturing systems (FMS) or develop a reliable computer-integrated manufacturing (CIM) system will depend on our ability to develop reliable and reusable softwares for large complex systems on a timely basis. To date, software design has not gone beyond the *ad hoc* trial-and-error stage. Consequently, the development of software is slow, expensive, unreliable, and unmanageable. The purpose of this paper is to provide a scientific basis for designing software. The approach used here is the axiomatic design, which is based on two design axioms: the Independence Axiom and the Information Axiom. The axiomatic approach is based on the recognition of the following common elements in design: the existence of independent domains (i.e., the consumer domain, the functional domain, the physical domain, and the process domain); the need to map between various domains during the design process; the decomposition of the characteristic vectors (i.e., functional requirements, design parameters, and process variables) in their respective domains; the zig-zagging required between the domains for decomposition; and the need to satisfy the design axioms during the design process. The axiomatic approach discussed in this paper provides decision making tools for software design in addition to systematic means of knowledge and data representation, synthesis and analysis of software, and the construction of the module-junction structure diagram.

Key words: Software, Design, Axioms.

Introduction

As computer-integrated manufacturing (CIM) is widely used by industrial firms, the reliability and cost of software will become increasingly important. This is a serious issue since the current state of software technology is still in the realm of an art, notwithstanding significant advances made in software technology in the form of SADT (1), YDM (2), Object-Oriented Software (3), and others (4, 5). As a result, the software development cost is high and even worse, the software maintenance cost is several times greater. Moreover, many softwares developed under government sponsorship cannot be used because they do not perform the intended functions. It is very difficult to change software, since one change affects many functions that should not be affected.

If the current state of software technology persists, the adoption of CIM will ultimately be limited by the cost, reliability, and flexibility of the software system rather than by the hardware cost(6). Therefore, it is imperative that systematic methodologies for software design be developed to systematize software development, to shorten the time for software production, to simplify the maintenance procedure, and to establish the rational foundation for software engineering. These objectives can only be achieved if the science base for software engineering can be established.

In this paper, a conceptual framework for software design is presented based on the design axioms (7). The axiomatic approach differs significantly from various methodologies used in software engineering in several important respects. First, it recognizes the existence of three or more design domains (e.g., consumer domain, functional domain, physical domain, and process domains) in software design. Second, the design process requires mapping between these domains. Third, each domain has a characteristic vector which can be decomposed to establish a hierarchical tree. Fourth, the decomposition process requires zig-zagging between the two adjacent domains. Finally, there are two design axioms which the mapping process must satisfy in order to create an acceptable software system. In short, the axiomatic design of software provides the conceptual frame, the criteria for acceptable software systems, and a methodology for software development.

In the past, these design axioms were applied primarily to the design of hardware, manufacturing processes, and organizations, although the applicability of the design axioms to software design was always assumed (7). The design axioms provide criteria for good designs, enable the selection of proper design parameters, the rapid evaluation of a large number of plausible ideas, and the selection of the best solution from among the acceptable options. The design axioms were applied to different kinds of problems, generating an impressive set of creative design solutions. This paper is the first of a series of papers which will be written on software design through the application of the design axioms.

Throughout the history of humanity, axioms have played significant roles in the development of mathematics (e.g., Euclidean geometry), and sciences (e.g., thermodynamics and mechanics). In this sense, the application of the design axioms is just another historical development. In the next section, the key concepts of the axiomatic approach to design will be briefly reviewed. A more comprehensive treatise of the axiomatic design is given by one of the authors (7, 8) elsewhere.

Review of the Key Concepts of Axiomatic Design

The design process consists of several steps:

1) Establishment of design goals to satisfy a given set of perceived needs;
2) Conceptualization of design solutions;
3) Analysis of the proposed solution;
4) Selection of the best design from among those proposed;
5) Implementation.

These activities occur between and in different design domains.

The design domains are illustrated in Figure 1. The consumer domain is where customer needs reside. These customer needs must be mapped into the functional domain where the customer needs are translated into a set of functional requirements (FRs), which constitute a characteristic vector. These FRs are then mapped into the physical domain, where the design parameters (DPs) are chosen to control/satisfy the FRs. The DPs in the physical domain are mapped into the process domain in terms of the process variables (PVs) In the case of software design, the process domains are in the form of subroutines, operating systems, compilers and so forth. Therefore, there can be many subdomains in the process domain, depending on the number of subroutines, and sub-subroutines. The term "space" is defined in this paper to represent subdomains. PVs in different spaces provide inputs to the physical domain.

The relationship between the domains is that the domain on the left is "*what we want*" and the domain on the right is "*How we will satisfy what we want*". Going from one domain to another is called mapping which is the synthesis phase of the design process.

Another important concept in axiomatic design is the fact that in each domain there exists a hierarchy. For example, in the functional domain, the FRs can be decomposed and formed into a hierarchy. However, the decomposition in each domain cannot be made independent of the hierarchies in other domains. In fact, in order to decompose a given FR at a given level of the FR hierarchy, we must first move over to the physical domain and conceive a design solution which is characterized by a set of DPs, and then move back to the functional domain and decompose the FR. This is illustrated in Figure 2. Therefore, the concept of decomposition and hierarchy is closely related to the concept of zig-zagging between domains to proceed with the decomposition from one level to the next lower level.

Finally, the most important concept in axiomatic design is the existence of the design axioms, which must be satisfied during the mapping process to come up with acceptable design solutions. The first design axiom is known as the **Independence Axiom** and the second axiom is known as the **Information Axiom**. They are stated as follows:

Axiom 1	The Independence Axiom Maintain the independence of functional requirements
Axiom 2	The Information Axiom Minimize the information content

The first axiom provides the criterion for acceptable design during the mapping process. The "product" design which is the mapping process between the functional domain and the physical domain can be represented by a design equation as:

$$\{FRs\} = [A]\{DPs\} \qquad (1)$$

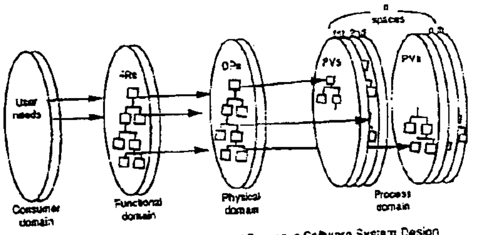


Fig. 1 Concept of Domain, Mapping and Spaces in Software System Design

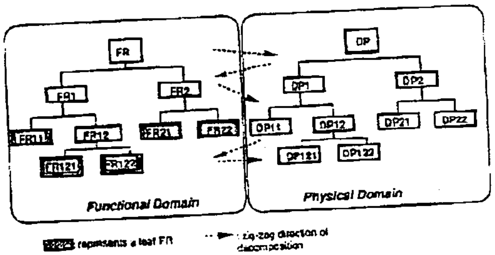


Fig.2 Hierarchical Tree Structures of Functional Requirements and Design Parameters.

This work was done at MIT during Sang-Gook Kim's sabbatical year.

where {FRs} is a vector that describes the functional requirements of the product in terms of its independent components FRi; {DPs} is a vector that describes the parameters that define the product in terms of its effect on {FRs}; [A] is a product design matrix. The elements of the product design matrix, Aij, are given by

$$A_{ij} = \frac{\partial FR_i}{\partial DP_j} \qquad (2)$$

which is a constant in linear design. In this case, the design window is infinitely large. In order to satisfy the Independence Axiom, [A] must be a diagonal or triangular matrix. The design that has a diagonal matrix is called an uncoupled design. When it is triangular, it is a decoupled design, which also satisfies the Independence Axiom, provided that DPs are changed in a specific sequence. All other designs are coupled designs.

A similar equation can be writen for mapping between the physical and the process domains as:

$$\{DPs\} = [B]\,\{PVs\} \qquad (3)$$

where [B] is a process design matrix and {PVs} are the vector that describe the process variables in the process domain. The elements of the [B] matrix are partial derivatives of DPs with respect to PVs. [B] matrix must be again either diagonal or triangular to satisfy the Independence Axiom. Equation (3) can be substituted in Eq. (2) to relate {FRs} to {PVs} directly.

The information content is defined in terms of the probability of successfully achieving FRs or DPs, depending on whether the design is for product or for process. Information is defined as

$$I_i = \log_2\left(\frac{1}{p}\right) \qquad (4)$$

where p is the probability of achieving the functional requirement FRi. When there are n FRs we must satisfy, the best design is the one with the least information content, i.e.,

$$I_{min} = \min\left\{\sum_{i=1}^{n} I_i\right\} \qquad (5)$$

In any design situation, the probability of success is given by what the designer wishes to achieve in terms of tolerance (i.e., design range), and what the system is capable of delivering (i.e., system range). As shown in Fig. 3, the overlap between the designer-specified "Design Range" and the system capability range, "System Range", is the region where the acceptable solution exists. Therefore, in the case of uniform probability distribution function, Eq. (4) may be written as:

$$I = \log_2\left(\frac{\text{System Range}}{\text{Common Range}}\right) \qquad (6)$$

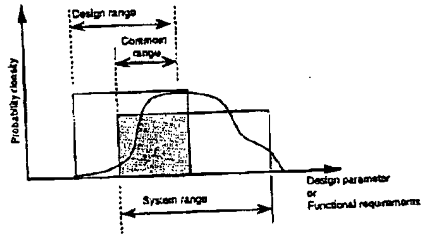


Fig. 3 Probability distribution of a system parameter. The curved line is for the case of non-uniform p-distribution. In the case of a "product design", the system parameter is the FR to be achieved, whereas in the case of a "process design", it is the DP which must be controlled through the proper choice of PVs. In the case of software design, PVs may be in the form of subroutines.

In design, there are always constraints. Constraints are similar to functional requirements at a given level of design hierarchy, except that they do not have to be independent of FRs or other constraints. Sometimes, constraints propagate from the process domain to the product domain to the functional domain.

These concepts will now be applied to software design.

Axiomatic Software System Design

Domains and Spaces

There are three or more domains in the axiomatic design world. The design procedure involves mapping and interlinking between the two adjacent domains in which the "*what we want*" and "*how to achieve what we want*" are stated. Software design is, as other product and process designs, a successive interplay between the *what* and *how* statements. Software design can be simplified by dividing the design issue in proper design domains, and by creating rational hierarchies for each characteristic vector through the use of of a continual mapping process.

The first step is to put the users' need of a software in the form of a set of FRs in the functional domain. Then the creative process sets in: a design solution must be conceived in the physical domain which must be characterized in terms of DPs. Once a set of DPs are chosen to satisfy the FRs, the acceptability of the solution can be checked in terms of the Independence Axiom. Some DPs can be further mapped into different spaces of the process domain following a similar procedure. For example, different subroutines can be developed in different spaces of the process domain, which define (or determine) the corresponding DPs. Different compilers and different operating systems may also reside in different spaces.

Hierarchical Structuring and Decomposition

The design of software system consists of an FR (Functional Requirement) hierarchy in the functional domain and a DP (Design Parameter) hierarchy in the physical domain. FRs are the outputs of a software and DPs are the key inputs to the software which can characterize/control the FRs. The software code is a set of design matrices which transform DPs to FRs at each level of the hierarchy. Therefore, the generation of the hierarchical trees of FRs and DPs of a software system constitutes the design process. This is done through mapping between the relevant domains.

The design process requires the successful transformation of "*what*" into "*how*" between the two hierarchical trees at each level. By zig-zagging the two domains, FRs and DPs are decomposed to develop the hierarchical tree structure. The mapping process begins with the definition of top-level FRs in the functional domain. A set of DPs are then selected in the physical domain to meet the specified FRs without violating the Independence Axiom. Ideation and analysis procedures are performed at each level of mapping from the FR to the DP domain. When the design solution cannot be finalized or completed by the selected set of DPs, the FRs need to be decomposed further. In defining the FRs at the next level lower, the DPs already selected will limit the allowable set of lower level FRs. The lower level FRs are a subset of many possible FRs, which are chosen within the constraints created when the specific DPs are selected. The following example shows how the decomposition process can be executed.

Consider a library software system, the task of which is to assign a call number to a new incoming book, update the keyword database, and process a search query without missing any single book which is relevant to the query. (Fig. 4)

STEP 1: FRs ---> DPs
The functional requirements (FRs) of the library software system are specified to meet the user needs as follows;

FR1 = Generate the DB (call number + keyword data base) for new incoming books
FR2 = Provide a list of references upon a search query using subject keywords only

At first, a DP1 is selected to fulfill the FR1. Then, a DP2 that satisfies the FR2 but does not affect the FR1 should be selected to create an uncoupled or at least decoupled design (8). An appropriate set of design parameters (DPs) that meet the FRs may be chosen as;

DP1 = Content of the book
DP2 = Set of subject keywords

The design matrix at this level is a triangular matrix which indicates that the design is a decoupled design, i.e.,

$$\begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \qquad (7)$$

where X represents a non-zero element, and O represents a zero element

STEP 2 & 3 : DPs ---> FRs
Since the relationship between FRs and DPs at the first level is not detailed enough to provide the desired output, the decomposition of the FRs is required. For the design defined by the selected set of the DP1 and DP2, the FR1 may be decomposed into FR11 and FR12 as

FR11 = Assign a call number (ID number) to a new book
FR12 = Generate subject keywords for the new book

FR2 is similarly decomposed as

FR21 = Find relevant reference books
FR22 = Generate a list of relevant references in response to a request

Step 4; FRs ---> DPs
DPs are now selected to satisfy the second level FRs, FR11 and FR12, in the same manner as explained in the step 1.

DP11 = Headline information of the book (field, title, author, publisher, year, etc.)
DP12 = The abstract of the book

In this example, it is assumed that the Library of Congress Classification (LCC) system has been used at the library, which should also be used for the new book. The field, title, authors and other headline information are the key inputs (DP11) to the call number system which can classify and assign a number to a new incoming book (FR11). Since the LCC system already exists as a subroutine in the designer's data base, the FR11 does not need to be decomposed any further.

The FR12 requires that the new book be registered in the keyword database for the future search purpose. The user typically does not have the exact information about the book being searched. Therefore, the user must generate a query with a series of most appropriate and commonly understood keywords which can properly represent the context of the book. Assuming that the abstract of the book has sufficient information, it is selected as the DP12. However, since the DP12 is not enough to describe the FR12, further decomposition is required which is shown in Step 6 and 7.

Since the call number assignment must be done before the keyword database is generated, the DP11 also affects the FR12. That is, the design is a decoupled design and its design matrix is triangular as

$$\begin{Bmatrix} FR11 \\ FR12 \end{Bmatrix} = \begin{bmatrix} X & O \\ X & X \end{bmatrix} \begin{Bmatrix} DP11 \\ DP12 \end{Bmatrix} \qquad (8)$$

STEP 5; FRs ---> DPs
Effective DPs that can control the FR21 and FR22 may be selected as

DP21 = Number of keywords which describe the field of interest
DP22 = String of keywords to find references which contain all of them

To process search query without missing any book which is relevant to the query (FR21), the query need to include as many subject keywords as possible. But, a greater number of keywords will result in a longer list of references, which will make the searcher very time-consuming. Therefore, a cross-indexing method of search using a string of keywords connected by OR, a Boolean operator, is adopted here, which selects the call numbers of books only in the common domain of the keywords when the keywords are connected by the OR operator. The cross-indexing method reduces the size of the reference list effectively, but may increase the possibility of missing relevant references.

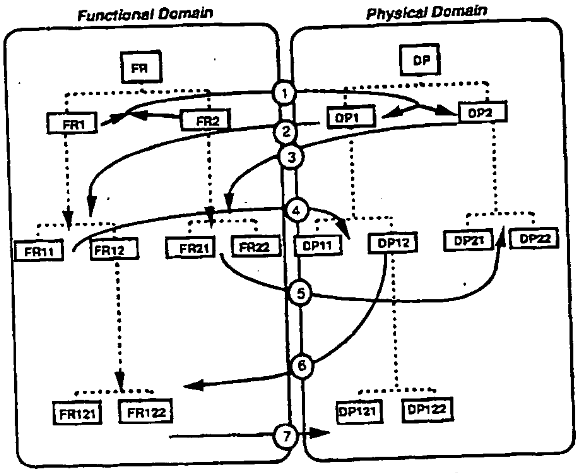


Fig. 4 The Zig-zag Decomposition of the FR and DP Hierarchy Structures
The solid lines show the zig-zagging path of decomposition, whereas the dotted lines indicate the vertical connection in the FR and DP hierarchies.

This design solution is a coupled one. It violates the Independence Axiom. Under normal design situation, one must go back to the "drawing board" and search for an uncoupled design solution. However, for the purpose of illustrating the use of the data flow diagram which is presented later in this paper, it is decided to use this design solution. The design equation for this coupled design may be written as

$$\begin{Bmatrix} FR21 \\ FR22 \end{Bmatrix} = \begin{bmatrix} X & X \\ X & X \end{bmatrix} \begin{Bmatrix} DP21 \\ DP22 \end{Bmatrix} \qquad (9)$$

Step 6 and 7;
The FR12 (i.e., generate keywords for the new book) is decomposed for the selected DP12 as

FR121 = Generate keywords for the book
FR122 = Generate a cross-indexing keyword database

The FR121 states that the keyword database should be rich enough not to miss any search query. This can be accomplished by putting every nouns in the abstract of a book to the keyword database, which may be chosen as the DP121. However, if this is done, the size of the database will grow enormously even with a single-paragraph abstract for each book. Therefore, the FR122 is defined so as to generate a cross-indexing keyword database, the size of which is as compact as possible. This may be accomplished by limiting the number of nouns extracted from each abstract. But it will adversely affect the FR121. If it is chosen as the DP122, this design solution would become a coupled design.

The DP122 which satisfies FR122 should not affect the FR121 according to the Independence Axiom. One way of doing it without affecting the FR121 is the use of the concept of synonym dictionary. The synonym dictionary is a new concept of data packing developed in the course of this example. It shrinks the size of the keyword data base by representing many number of nouns extracted from the abstract with a small set of key nouns. This synonym-based keyword data base can be used to retrieve the desired book by means of the synonym dictionary to generate a new set of keywords before beginning the search. Then, the selected set of DPs may be stated as

DP121 = Nouns extracted from the abstract. (trivial ones eliminated)
DP122 = Synonym dictionary

For the purpose of the illustration, we will assume the synonym dictionary is so powerful that the keyword data base can become sufficiently compact. Then, the design matrix is diagonal representing an uncoupled design as:

$$\begin{Bmatrix} FR121 \\ FR122 \end{Bmatrix} = \begin{bmatrix} X & O \\ O & X \end{bmatrix} \begin{Bmatrix} DP121 \\ DP122 \end{Bmatrix} \qquad (10)$$

Termination of Decomposition
When the decomposition process propagates down to the lower levels, a designer can reach the level where one or more FRs can be fully satisfied (or controlled) by the selected set of DPs. If the FRs need not be decomposed any further, they form terminal nodes of the hierarchical tree. The decomposition process terminates when all the branches of the FR tree form the terminal nodes. A terminal node is defined in this paper as a leaf of the FR tree (see Fig. 2).

A module is defined in this study as a block of software which generates an output (physical data form of an FR leaf) from a set of DPs selected. Each FR leaf has one module of software which transforms a set of inputs to the output. For example, the module, M121 in Fig. 5 transforms the DP121 to the FR121.

Based on this process of decomposition and development of uncoupled or decoupled designs, axiomatic design inherently assures good modularity which has been a concern of many CASE (Computer Aided Software Engineering) research groups (5), because the modules are generated to make the associated FRs uncoupled or decoupled. Furthermore they contain less information than coupled designs. A module can be readily coded into a program with a proper set of DPs at this stage of design.

Module-Junction Structure Diagram
Fully decomposed hierarchical trees of FRs and DPs contain sufficient information to build the software system. Modules for FR leaves are coded into programs and can be constructed into a system to function as specified by the consumer. This is the system integration phase of software design.

In order to facilitate the system integration and the immediate access to current CASE users, the hierarchical tree structures of FRs and DPs and the analysis results in the form of design matrix, are represented in a single diagram (Fig. 5). It will be defined as the "Module-Junction Structure Diagram". It is composed of modules and junctions which represents FR leaves and their vertical integration, respectively. For example, the FR12 can be satisfied by having both the FR121 and FR122 satisfied. This brings the question of "How should the outputs of M121 and M122 be put together to generate the FR12?". This can be accomplished by utilizing the readily available analysis result which were used in decomposing the FR12 into FR121 and FR122. This is defined as a junction which vertically integrates child modules into a parent module.

There are three types of junctions in the junction structure diagram; summation (S), control (C), and feedback (F) junctions (Fig. 7). When the child FRs are uncoupled, the parent (but single) FR is satisfied by combining all the outputs of its child modules; this is the summation junction (not an arithmetic summation). When they are decoupled, the parent FR will use the output of the left-hand side module to control the execution of the right-hand side modules. This is the control junction. The coupled junction feedbacks the output of the right-hand side modules to the left, which requires number of iterations in processing data or in use of the program. It should be noted that the coupled junction must be avoided, especially during the decomposition process. When there are many sub-modules included in the feedback junction, the program will quickly become unmanageable.

A main module is defined as that which contains all the junctional properties at each level (Fig. 6). A software system has one main module and n modules corresponding to n FR leaves.

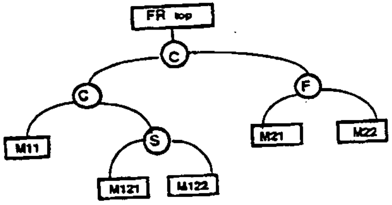


Fig.5 Module-Junction Structure Diagram

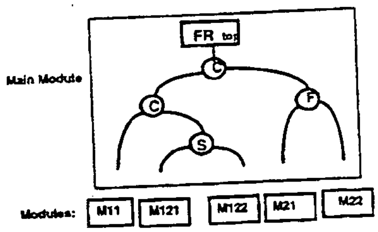


Fig. 6 Modules and the Main Module

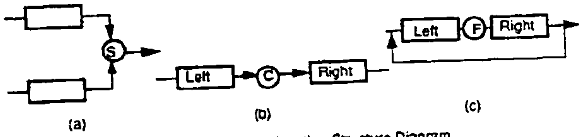


Fig. 7 Junction Properties of the Module-Junction Structure Diagram
(a) Summing Junction (Uncoupled Case) (b) Control Junction (Decoupled Case)
(c) Feedback Junction (Coupled Case)
In the cases of control and feedback junctions, the information should flow from the left-hand module to the right-hand module, whereas in the case of summing junction, the chronological order of the information flow is not important.

Data Flow Diagram
By using the junction characteristics, the module-junction structure diagram can be easily converted to a network-type diagram which shows the flow of data stream among modules (Fig. 8). This is similar to the data flow diagram which has been the key graphical representation in conventional structured analysis and design methods (10). The result of axiomatic design should be the starting point to the programmers who can code programs for each module, all of which then can be structured to develop the main module.

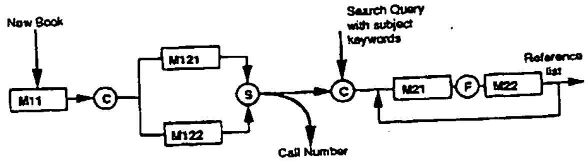


Fig.8 Data Flow Diagram which is derived from the Module-Junction Structure Diagram

Design of a Rib Design Software -- A Case Study

The steps involved in axiomatic approach to software design has been described in preceding sections. In order to illustrate the concepts of axiomatic design further using a manufacturing related example, the design of software system for injection molding of plastic parts will be described in this section.

Consider the design of a injection molded part. The primary shape of an injection molded part is usually designed by an application engineer to satisfy a set of functional requirements, including the desired external appearance. The moldability of the part and its mechanical performance are considered separately from the product design and need to be provided by a mold designer. Supplementary features are then added to the primary shape by experienced mold engineers to reinforce the structure, to facilitate the melt flow and/or to supplement the functional requirements with additional FRs. Ribs and bosses are the typical supplementary features being added to the primary shape of injection molded parts.

The major role of a rib structure is to enhance the structural rigidity of a plastic part while maintaining the uniform wall thickness and minimizing the cycle time of the molding process. A rib can also play a role of a secondary runner which quickly delivers the polymer melt to the remote region where the slow filling may cause defects such as short shots, flow marks, and voids. A bad design of a rib structure may deteriorate the quality of the product by causing sinkmarks, warpage, short-shot and difficulty in removing the part from the mold. Therefore, the design task for a rib structure consists of the specification of reinforcing requirements, the choice of shape parameters to meet the reinforcing requirements and moldability, and the check of the procedure for making defect-free parts (9). A sofware system needs to be developed which can assist mold designers to make the primary shape of a plastic part moldable and mechanically sound.

Top Level of the FR and DP hierarchies

These consumer requirements can be stated as the FRs in the functional domain. The highest level FRs are:

FR1 = Specify reinforcement requirement
FR2 = Generate supplementary geometry
FR3 = Check the performance of the reinforced structure by simulation

The DPs which make the FRs decoupled are selected as

DP1 = Primary shape of the plastic part designed by an application engineer (cavity side)
DP2 = Ribbed structure (core side)
DP3 = Applied loads (expected)

The design matrix at this level is a triangular matrix which indicates that the design solution is decoupled, i.e.,

$$\begin{Bmatrix} FR1 \\ FR2 \\ FR3 \end{Bmatrix} = \begin{bmatrix} X & O & O \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \\ DP3 \end{Bmatrix} \qquad (11)$$

At this level, it is recognized that once the DP1 and DP2 are fixed, the FR3 can be readily met by a subroutine which is a commercially available structural analysis software package. (ANSYS is selected in this case study.) Then the FR3 becomes an FR leaf in the functional domain and the subroutine constitutes the module, M3.

The FR1 and the FR2 need to decomposed further since the design solution can not be finalized by the selected DPs.

Second Level

In order to decide whether the reinforcement is required or not, the structural performance of the given primary geometry under certain loading conditions must be quantified. However, the loading conditions of most plastic parts are not pre-specified explicitly, but are given such as to avoid the allowable maximum deflection of the lid plate or the possible stress concentration at corners and handles, and so forth. Therefore, the use of a complicated finite-element program for the complete structural analysis is not an effective method due to the laborious data preparation effort and enormous computation time.

The FR1 is decomposed as follows;

FR11 = Characterize the structural performance of a primary geometry under implicit loading conditions
FR12 = Calculate the reinforcement requirement

Depending upon the shape of the part, the regions where the maximum deflection or stress concentration occurs are identified by the designer intuitively; for example, the wide bottom plate or the long side edge of the part, which can be simplified to elementary geometries. Likewise, the complex shape of the primary geometry can be characterized by a set of elementary geometries where simple structural formulas for thin plates and beams can be applied as the initial guess for the reinforcing requirements. In this case study, three elementary geometries for the rib reinforcement are formulated, such as; a rectangular plate, a circular plate, and a curved beam to which appropriate structural formulas from handbook can be applied respectively (18). The selected DPs are:

DP11 = Elementary geometries
DP12 = Structural formulas from the handbook

The resulting design matrix becomes a triangular one.

The FR2 is decomposed as

FR21 = Generate ribs which do not deteriorate the manufacturability of the part
FR22 = Generate ribs which can meet the reinforcing requirement

The FR21 can only be satisfied by having both the DPs in the physical domain and the PVs in the process domain not to violate the Independence Axiom. Assuming that the PVs will be selected appropriately, the cross-sectional shape of the rib is the key input which controls the FR21. The number of ribs and their locations should be determined to meet the reinforcing requirements after the cross-sectional parameters are determined. This can be accomplished by determining the moment of inertia of the ribbed structure which is composed of *n* number of ribs of certain cross-sectional shape to provide an equivalent stiffness to that of the reinforced structure of flat thickness. In this way, the FRs can be decoupled. The DPs that can control FR21 and FR22 are selected as

DP21 = Cross-sectional shape of a rib structure
DP22 = Number of ribs

At this level, it seems that the FR11, FR12, and FR22 can be described by the selected set of DPs. These FRs form the terminal nodes of the FR tree and the software modules for the FR leaves are developed accordingly. On the other hand, the FR21 needs to be decomposed further since the DP21 is not sufficient to describe it.

Third Level

The manufacturability of ribs can be specified so as to assure ejectability, avoid excessive warpage, and minimize the size of a sinkmark which is a local dent formed on the surface opposite to the rib. The FR21 is decomposed as

FR211 = Avoid warpage
FR212 = Assure ejectability
FR213 = Minimize sinkmarks

Cross-sectional parameters of a rib should be determined based on the process characteristics of injection molding. Cross-sectional parameters include the height, the root thickness, the filleting radius, and the draft angle. The wall thickness and the material to be injected are predetermined together with the primary geometry, and will act as constraints. For a given material and wall thickness, the cross-sectional parameters have proper ranges in order to prevent jamming, warping, or sinkmark formation. By considering the major causality between the defects and cross-sectional parameters, the following DPs are selected.

DP211 = Rib height
DP212 = Draft angle
DP213 = Root thickness

However, the design matrix becomes a coupled one if the allowable warpage (given in terms of tolerances) cannot be satisfied by the process as

$$\begin{Bmatrix} FR211 \\ FR212 \\ FR213 \end{Bmatrix} = \begin{bmatrix} X & O & X \\ X & X & O \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP211 \\ DP212 \\ DP213 \end{Bmatrix} \qquad (12)$$

Injection molding is inherently a coupled process between the flow and the solidification of polymer melts (10). Therefore, it is very difficult to select DPs which can make the design matrix uncoupled or decoupled, unless the tolerance is very large. This is an example that the decomposition process may include zig-zagging not only between the FR and DP domains, but also between the DP and PV domains in order to meet the FRs in relation with manufacturability issues. One way of handling this kind of problem (design for manufacture) is the use of the knowledge-based system. The constraints and expert knowledge of determining the cross-sectional parameters between the DPs in the physical domain and process variables in the PV domain are encoded as rules in the expert system module, RIBBER (17). Then the decision of the cross-sectional parameters is to be made through the interaction between the designer and RIBBER. The expert system may give explanations on the causal defects, and warnings and advices to the designer to generate acceptable cross-sectional geometries of ribs.

Module-Junction Structure

As a result of the zig-zag decomposition, hierarchical structures of FRs and DPs are constructed as the part of the axiomatic design. The terminal nodes are formed when the FRs can be satisfied by the selected DPs. A module for each FR leaf can be generated independently as long as the FRs at the same branch are not coupled. Figure 9 shows the module-junction structure diagram of the rib design software which represents both the hierarchical structures and design matrices which are essential to integrate the individually developed modules vertically.

It can be noted that the use of the expert system encapsulates the modules with the feed-back junction, which otherwise will be laborious to develop and also interfere the development of other modules due to their coupled nature.

A more detail report of the development of a rib design software is given by one of the authors elsewhere (9).

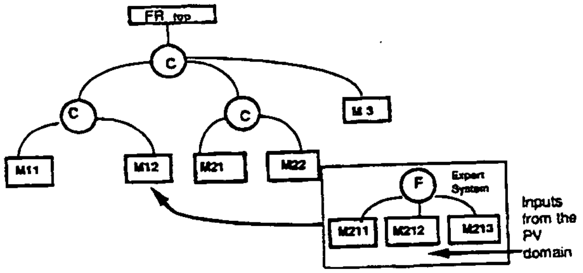


Fig. 9 The Module-Junction Structure Diagram for the Rib Design Software

Axiomatic Design Method v.s. Structured Design Methods

Software system design is a continual mapping process between various domains shown in Fig. 1. It involves specification and decomposition of the functional requirements and design parameters starting from highest level to the lowest level FRs and DPs, including the specifications and constraints from which program code can be routinely generated. Developing a set of correct FRs (or system specifications) is the pre-requisite for sound software development because errors made at this stage affect all subsequent decisions and thus, are difficult to correct (11). This is one of the shortcomings of the current empirical practice, since they often lack rigor in specifying requirements of complex systems (14).

Since the mid 1970s, much progress has been made in software design through the development of structured technologies which systematize the analysis and design of software systems. Structured analysis and structured design are the most popular CASE (Computer Aided Software Engineering) techniques. It is intended to bring structured approach to decision making in requirement analysis and design. SADT (1), DeMarco Analysis Methodology (12), and Gane and Sarson Methodology (13) are some of the well known structured analysis methods for requirement specification, whereas Yordon Design Methodology (16), Jackson Design Methodology (17), Warnier-Orr Methodology (18) are commonly cited structured design methods. These methodologies consists of a set of representation techniques including data flow diagrams, entity-relationship diagrams, and structure charts.

Structured analysis methods use a top-down functional decomposition method to define system functional requirements. Structured design methods are created to provide a step-by-step procedure for developing, documenting and evaluating designs (16). All structured techniques have similar underlying concepts which may be stated as follows (14):

- Top-down, hierarchical structuring
- Divide and conquer
- Graphic communication and documentation tools

However, the structured analysis and design methods do not provide decision making criteria as to what constitutes a good design. Consequently, they cannot direct the designer to the optimal design solution in a systematic and scientific manner. Therefore, the use of those structured techniques is still confined to the inter-level consistency checking and user-friendly graphical operations for bookkeeping purposes. Clearly, the major weakness of these methodologies is the absence of fundamental principles and criteria for decision making. Furthermore, the software development is rendered to be unnecessarily complex by not recognizing the existence of distinct domains and the need to map between different domains. These concepts are essential in dealing with complex problems.

SADT (1) has been one of the well known structured analysis and design methodologies. The SA box is the basic idea-entity representing tool of SADT. The four sides of the SA box mean input, output, control, and mechanism as shown in Fig. 10 (a). It is supposed that input is transformed into output under control. However, the four sides of SA box may become very much complicated in respect of axiomatic design as shown in Fig. 10 (b). The "input" may include feedbacks from the outputs of downstream boxes, which is a coupled design and therefore, should be avoided. The "mechanism" of SADT may represent the input from a subroutine to the module which must be handled in the process domain. The "control" input may be the data flow via the control junction which is the result of a decoupled design as shown in Fig. 7 and 8. Although the decoupled design satisfies the independence axiom, its early introduction at the high level of the FR hierarchy complicates the subsequent design procedure. That is, the design process will become more and more complicated as the decomposition proceeds further. Decoupled designs cannot be handled effectively by SADT since it does not have fundamental principles, criteria, and concept of distinct domains in the process of design.

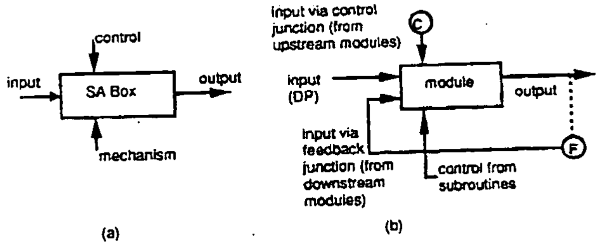

control
input
SA Box
output
mechanism
(a)
input via control junction (from upstream modules)
C
input (DP)
module
output
input via feedback junction (from downstream modules)
F
control from subroutines
(b)

Fig. 10 Comparison of the SADT and Axiomatic Approach of Software Design
(a) Structural Analysis Box (b) Module-Junction Structure

Axiomatic design is based on the concept that there exists fundamental principles in design. Therefore, it provides a unique design methodology based on the absolute referent (i.e., design axioms) for the synthesis and analysis of design (7). It provides design evaluation criteria, which enable the designer to eliminate poor design solutions at every stage of decision-making and search for new ideas, promptly. They also enable the designer to select the best among those proposed. It incorporates the concept of structured hierarchy, not only in the functional domain but also in the physical and process domains (8).

An Ideal Software System

The Independence Axiom states that at a given level of the FR hierarchy shown in Fig. 2, all the FRs must be maintained independent when DPs are changed to alter their respective FRs. This fact can be used to create a Thinking Design Machine for products and processes (7, 8). This same concept can be used to develop a software design system that will considerably simplify the software design process.

For this purpose, consider a functional requirement hierarchy, where the lowest level FR^Ls (sometimes called "leaves") do not require further decomposition. The upper level FRs can be constructed by means of FR^Ls. Then, we can construct a software module data base as follows:

$$FR^L_1 = M(DP^1_{1,} DP^2_{1,} DP^3_{1,} \ldots . DP^n_{1.})$$
$$FR^L_2 = M(DP^1_{2,} DP^2_{2,} DP^3_{2,} \ldots . DP^m_{2.})$$
$$\vdots$$
$$FR^L_x = M(DP^1_{x,} DP^2_{x,} DP^3_{x,} \ldots . DP^p_x) \qquad (12)$$

Equations (12) are a library of softwares (i.e., data base). For example, the first equation for FR^L_1 states that any one of the modules (e.g., $DP^1_{1,} DP^2_{1,} DP^3_{1,} DP^{10}_{1,}$ etc.) in the database can be used to control FR^L_1. Similarly, any one of the $DP^m_{2,}$ can satisfy FR^L_2.

Now suppose we need to develop a software for a problem consisting of the following three leaf FRs: FR^L_1, FR^L_3, FR^L_x. The best software package is the one that consists of DPs which affect only one of the FR^Ls. This can be expressed as

$$\begin{Bmatrix} FR^L_1 \\ FR^L_3 \\ FR^L_x \end{Bmatrix} = \begin{bmatrix} X & O & O \\ O & X & O \\ O & O & X \end{bmatrix} \begin{Bmatrix} DP^{25}_1 \\ DP^{18}_3 \\ DP^7_x \end{Bmatrix} \qquad (13)$$

In conceptualizing the design solution represented by Eq. (13), only those DPs that affect only one of the FRs are chosen. The above software package is an uncoupled design which satisfies the Independence Axiom.

When we cannot find an uncoupled design, we can look for a decoupled solution which also satisfies the Independence Axiom. For example, when we first look for a DP that can yield FR^L_1, it can be anything in the data base; for example, DP^{25}_1. However, when we then look for the software module that can yield FR^L_3 after selecting DP^{25}_1, we must look for a DP^y_3 that does not affect FR^L_1. Finally, when we look for a DP^j_x that can satisfy FR^L_x after FR^L_1 and FR^L_3 are taken care of, there are now two conditions: the DP^j_x in the data base that can satisfy the Independence Axiom is the one that does not affect both FR^L_1 and FR^L_3. Then, the design equation for the decoupled software package can be written as:

$$\begin{Bmatrix} FR^L_1 \\ FR^L_3 \\ FR^L_x \end{Bmatrix} = \begin{bmatrix} X & O & O \\ ? & X & O \\ ? & ? & X \end{bmatrix} \begin{Bmatrix} DP^{25}_1 \\ DP^{87}_3 \\ DP^{28}_x \end{Bmatrix} \qquad (14)$$

The question marks indicate that even if the design elements indicated by (?) are not zeros, the software will execute the computation as long as the modules are computed in the sequence indicated, i.e., DP^{25}_1 first, then DP^{87}_3, and finally followed by DP^{28}_x. Such a program will involve many control junctions shown in Fig. 7.

If the data base is given for FRs at higher levels, the software can be constructed without regard to the specific level of the FR hierarchy. On the other hand, if the higher level FRs can be constructed from the elementary level FR^Ls through the vertical integration, then, we do not need to have these higher level library of software packages.

The key to the creation of an idealized software package is the creation of the complete data (i.e., module) base.

In many situations, the data base is so extensive that we may be able to find another set of DPs which satisfy FR^L_1, FR^L_3, FR^L_x independently. Then, the question becomes which set is a better solution for the given set of functional requirements.

In order to decide which is a better solution, we have to invoke the Information Axiom. FR^Ls may have tolerances associated with them. However, different software modules may yield the results for FR^L with different degrees of accuracy. Therefore, the information contents can be computed for each FR^Ls. The best solution, according to the second Axiom, is the one with the minimum information content.

Summary

In this paper, a conceptual framework for software design is presented based on the design axioms. The axiomatic approach for software engineering can be characterized in terms of following:

1) The software design begins with the specification of FRs from the perceived needs of the user.

2) Axiomatic approach recognizes the existence of the independent domains and spaces in software design.

3) The design process requires mapping between these domains.

4) There are two design axioms that must be satisfied by the mapping process in order to develop an acceptable software system.

5) Each domain has a characteristic vector which can be decomposed to establish a hierarchical tree. In software design, the outputs constitute the FR tree in the functional domain, while key inputs form the DP tree in the physical domain.

6) The decomposition process requires zig-zagging between the two adjacent domains.

7) The decomposition process terminates when all the branches of the FR hierarchical tree (FR leaves) can be fully satisfied by the selected set of DPs.

8) Each FR leaf has one module of software which is the design matrix between the FR leaf and the selected DPs.

9) Axiomatic design inherently assures the modularity of the software system since modules are generated to make FRs uncoupled or decoupled.

10) The module-junction structure diagram is devised to represent the vertical integration of the modules. Three types of junctions are defined based on the coupling property of the design matrix, such as; summation, control and feedback junctions.

The axiomatic design of software provides the conceptual frame, the principles for software systems, the criteria for acceptable software systems, and a methodology for software development. The concept of the Thinking Design Machine as a tool for software development is established based on these frame, criteria, and methodologies.

Dedication

One of the authors of this paper, Sun-Jae Kim, was a graduate student at MIT, working actively on the project called Thinking Design Machine. He passed away unexpectedly. He was a man with a great promise. His untimely departure from this world is a loss to engineering, industry, and humanity at large.

References

[1] Ross, D.T., "Application and Extensions of SADT", IEEE Computer, April 1985, pp. 25-35.

[2] Bjorner, D., "On the Use of Formal Methods in Software Development", Proceedings of the 9th International Conference on Software Engineering, Computer Society Press, Los Alamitas, CA, 1981, pp. 17-29.

[3] Wifts-Brock, R., Wilkerson, B., and Wiener, L., Designing Object-Oriented Software, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[4] Dart, S., Ellison, R.J., Feiler, P.H., and Habermann, A.N., "Software Development Environments", Computer-Aided Software Engineering", IEEE Computer Society Press Technology Series, (ed., V. Vemuri), 1989, pp. 17-27.

[5] Karimi, J., and Konsynski, B.R., "An Automated Software Design Assistant", ibid., pp. 65-81.

[6] Bjorke, O, "Software Production - The Bottleneck of Future Manufacturing System," Annals of the CIRP, Vol.24/2, 1975

[7] Suh, N.P., The Principles of Design, Oxford University Press, New York, 1990.

[8] Suh, N.P. and Sekimoto, S., "Design of Thinking Design Machine", Annals of the CIRP, Vol. 39/1, 1990.

[9] Huh, Y.J. and Kim, S.G., "A Knowledge-based CAD System for Concurrent Product Design in Injection Molding", Intl. J. of CIM, accepted for publication

[10] Kim, S.G. and Suh, N.P.,"The Knowledge-based Synthesis System for Injection Molding", Intl. J. of Robotics and CIM, Vol. 3, No. 2, 1987

[11] Boehm,B, McClearn, R., and Unfrig, D.,"Some Experiences with Automated Aids to the Design of Large-Scale Reliable Software," IEEE Trans. on Software Eng., 1, No.1, 1975

[12] DeMarco, T.,Structured Analysis and System Specification, Yourdon, Inc., New York, 1978

[13] Gane, C. and Sarson, T., Structured Systems Analysis: Tools and Techniques, IST Databooks, New York, 1977

[14] Martin, J. and McClure, C, Structured Techniques for Computing, Prentice-Hall, New Jersey, 1985

[15] Yourdon, E.,"The Emergence of Structured Analysis," Computer Decisions, 8, No.4, 1976

[16] Yourdon, E. and Constantine, L., Structured Design, Prentice-Hall, NJ, 1979

[17] Jackson, M.A., Principles of Program Design, Academic Press, New York, 1975

[18] Orr, K., Structured Systems Development, Yordon Press, New York, 1977

[19] Roark, R.J., Formulas for Stress and Strain, McGraw Hill, 1965